Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 5

Тема: Основы работы с коллекциями: итераторы

Студент: Пивницкий Даниэль Сергеевич
Группа: 80-206
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Разработать шаблоны классов согласно варианту задания.
Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат.
Классы должны иметь публичные поля. Фигуры являются фигурами вращения, т.е. равносторонними (кроме трапеции и прямоугольника).
Для хранения координат фигур необходимо использовать шаблон std::pair.

2. Описание программы

Основу программы составляет шаблонный класс queue, являющийся реализацией структуры “очередь”. Для стека определены необходимые методы вставки и удаления. Создан итератор с возможностью работы со стандартными алгоритмами. В функции main происходит обработка тестов. Также реализованы дополнительные структуры и классы в соответствии с вариантом задания.

3. Набор тестов

4. Результаты выполнения тестов

5. Листинг программы

```
//
//  main.cpp
//  lab5
//  Variant 19
//  М8о-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
/*
Разработать шаблоны классов согласно варианту задания.
Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат.
Классы должны иметь публичные поля. Фигуры являются фигурами вращения, т.е. равносторонними (кроме трапеции и прямоугольника).
Для хранения координат фигур необходимо использовать шаблон  std::pair.
*/
#include <iostream>
#include <memory> // unique_ptr
#include <algorithm> // for_each, count_if
#include "rectangle.hpp"
#include "queue.hpp"

using figure_type = int;
```

```
void writeMenu() {
	std::cout << "0\tВыход из программы\n";
	std::cout << "1\tЗапрос меню\n";
	std::cout << "2\tДобавить фигуру в очередь\n";
	std::cout << "3\tУдалить фигуру из очереди по номеру\n";
	std::cout << "4\tПечать фигур в очереди\n";
	std::cout << "5\tПечать количества фигур, площадь которых меньше заданной\n";
}

int main() {
	Queue <Rectangle<figure_type>> st;

	short comand;
	bool menu=1;

	writeMenu();
	while(menu) {

		std::cout << '\n';
		std::cout << "Введите команду: ";
		std::cin >> comand;
		std::cout << std::endl;

		switch(comand)
		{
			case 0:
				menu = 0;
				break;
			case 1:
				writeMenu();
				break;
			case 2:
				{
					std::unique_ptr<Rectangle<figure_type>> rec = std::make_unique<Rectangle<figure_type>>();
					try {
						std::cin >> rec;
						st.Push(std::move(rec));
					} catch(const char* exception) {
						std::cerr << "ERROR: " << exception << '\n';
					}

				}
				break;
			case 3:
				{
					unsigned int id;
					std::cout << "Введите индекс элемента, который
```

```
нужно удалить: ";
                                std::cin >> id;
                                std::cout << std::endl;
                                try {
                                      st.Remove(id);
                                } catch(const char* exception) {
                                      std::cerr << "ERROR: " << exception <<
'\n';
                                }

                          }
                          break;
                    case 4:
                          std::for_each(st.begin(),                   st.end(),
print<figure_type>);
                          break;
                    case 5:
                          {
                                int max=0;
                                std::cout << "Введите площадь: ";
                                std::cin >> max;
                                std::cout << std::endl;
                                size_t a = std::count_if(st.begin(), st.end(),
[max](std::unique_ptr<Rectangle<figure_type>>&                    elem){return
elem->Area()<max;});
                                std::cout << a << std::endl;
                          }
                          break;

             }
      }

      return 0;
}



//
//  queue.hpp
//  lab5
//  Variant 19
//  М8о-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#include <iostream>
#include <memory> // unique_ptr
#pragma once
using namespace std;
```

```
template <class T>
struct Queue_iterator {

    using figure = T;
    using iterator_category = forward_iterator_tag;
    using value_type = unique_ptr<T>;
    using pointer = unique_ptr<T>*;
    using reference = unique_ptr<T>&;
    using difference_type = unsigned int;

    pointer it;

    Queue_iterator(){}
    Queue_iterator(pointer queue): it(queue) {}

    void next() {
        it+=1;
    }

    reference operator*() {
        return *it;
    }

    Queue_iterator<figure>& operator++() {
        it+=1;
        return *this;
    }

    Queue_iterator<figure>& operator+(int a) {
        it+=a;
        return *this;
    }

    Queue_iterator<figure>& operator-(int a) {
        it-=a;
        return *this;
    }

    bool operator!=(const Queue_iterator& rvl) {
        return it != rvl.it;
    }

};

template <class T>
struct Queue {

    using figure = T;
    using iterator = Queue_iterator<T>;
    using value_type = unique_ptr<T>;
    using pointer = unique_ptr<T>*;
```

```
unsigned int size;
unsigned int buffer;
pointer queue;

Queue(): size(0), buffer(0), queue(NULL) {}
~Queue() {
    delete []queue;
}

void Push(value_type elem) {
    if (buffer == 0) {
        queue = new value_type[1];
        buffer = 1;
    }
    if (size == buffer) {
        value_type *temp = new value_type[buffer << 1];
        buffer<<=1;
        for (unsigned int i = 0; i < size; ++i) {
            temp[i] = move(queue[i]);
        }
        delete []queue;
        queue = temp;
    }
    queue[size] = move(elem);
    ++size;
}

void Pop() {
    if (size==0) return;
    if (size>1) {
        for (unsigned int i = 0; i < size-1; ++i) {
            queue[i] = move(queue[i+1]);
        }
    }
    --size;
    if (size*2 < buffer) {
        value_type *temp = new value_type[buffer >> 1];
        buffer>>=1;
        for (unsigned int i = 0; i < size; ++i) {
            temp[i] = move(queue[i]);
        }
        delete []queue;
        queue = temp;
    }
}

void Remove(unsigned int id) {
    if (id >= size) throw "Индекс больше размера очереди!";
    for (unsigned int i=0; i<id; ++i) {
        value_type s = Top();
```

```
            Pop();
            Push(move(s));
        }
        Pop();
        for (unsigned int i = 0; i < size - id; ++i) {
            Push(Top());
            Pop();
        }
    }

    value_type Top() {
        if (size>0) {
            return make_unique<figure>(*queue[0]);
        } else {
            throw "Очередь пуста!";
        }
    }

    void insert(iterator it, value_type elem) {
        *it = elem;
    }

    void erase(iterator it) {
        while (it+1 != end()) {
            *it = *(it+1);
            it++;
        }
        size--;
    }

    iterator begin() {
        return iterator(queue);
    }

    iterator end() {
        return iterator(queue + size);
    }
};



//
//  rectangle.hpp
//  lab5
//  Variant 19
//  М8о-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#include <iostream>
```

```
#pragma once
using namespace std;

template <class T>
struct Rectangle {
    pair<T,T> point;
    T height;
    T width;

    Rectangle(): point{0,0}, height(0), width(0){}
    Rectangle(T x, T y, T h, T w): point{x,y} {
        if (h<0 or w<0) throw "Ширина и высота должны быть положительными!";
        height = h;
        width = w;
    }

    T Area() { return height*width; }
};

template <class T>
ostream& operator<< (ostream& out, const unique_ptr<Rectangle<T>> &A) {
     out << "Точки прямоугольника: " << "(" << A->point.first << ", " <<
A->point.second << ") "
        << "(" << A->point.first+A->width << ", " << A->point.second << ") "
                    <<  "("  <<  A->point.first+A->width  <<  ",  "  <<
A->point.second-A->height << ") "
        << "(" << A->point.first<< ", " << A->point.second-A->height << ") "
        << "\n";
    return out;
}

template <class T>
istream& operator>> (istream& in, unique_ptr<Rectangle<T>> &A) {
    cout << "Введите координаты положения прямоугольника: ";
    in >> A->point.first >> A->point.second;
    cout << "Введите ширину прямоугольника: ";
    in >> A->width;
    cout << "Введите высоту прямоугольника: ";
    in >> A->height;
    if ((A->width < 0) || (A->height < 0)) throw "Ширина и высота должны быть
положительными!";
    return in;
}

template <class T>
void print(std::unique_ptr<Rectangle<T>>& elem) {
    std::cout << elem;
}
```

6. Выводы

В процессе выполнения лабораторной работы освоил основы работы с умными указателями в C++, а также повторил базовую реализацию некоторых коллекций языка.

7. Список литературы

1.Руководство по написанию кода на C++ [Электронный ресурс].
URL:https://metanit.com/cpp/tutorial/
Дата обращения: 10.09.2019

2.Документация по C++ [Электронный ресурс]. URL:
https://docs.microsoft.com/ru-ru/cpp
Дата обращения 12.09.2019